Fine Hierarchies of Generic Computation *

Jerzy Tyszkiewicz

Mathematische Grundlagen der Informatik,
RWTH Aachen, Ahornstraße 55,
D-52074 Aachen, Germany.
jurek@informatik.rwth-aachen.de

Abstract. Suppose that you are a user of a commercial relational database, accessible over the Internet, whose owner has decided to copy the price lists of the US telephone companies — first order queries are for free just like local calls, because they are local by the theorem of Gaifman [6]. All recursive queries, being potentially non-local, are charged, for simplicity let us assume \$1.00 for a Boolean query. Non-Boolean queries are certainly not allowed, because the user would require all the data to be sent to him by issuing the first order identity query, and then manipulate with it himself, without any pay.
These are the rules. Well, what is your strategy, to compute all you want to know about the database, paying as little as possible? And how much will the total price be?
We answer this question, showing that the question whether you can get your answer without any costs at all, depends on whether or not the theories of databases in the provided query language are finitely axiomatizable.
Thus, assuming there is a limit on the number of variables allowed in queries, if the database query language is the fixpoint logic, you can get everything for free. When it is Datalog however, even with inequality and negation of the edb's, you have to pay. We present a method, which for graphs of n vertices costs about \$ $\log_2 \log_2 n$. Thus querying a graph with 1 Terabyte vertices costs \$7.00. We demonstrate that this price cannot be substantially reduced without causing a large computational overhead.

1 Introduction

At the heart of the story in the abstract of this paper is the question:

Is it possible to evaluate recursive queries in nonrecursive query languages, perhaps by means of evaluating many nonrecursive ones and analyzing such obtained data? I.e., is it possible to compensate for the lack of recursion in the queries by computational power outside of the query language?

We attempt to characterize query languages in which it is possible, and determine how many recursive queries are really necessary, if it is impossible.

* This research has been supported by a Polish KBN grant 8 T11C 002 11 and by the German Science Foundation DFG.

1.1 A longer explanation of the problem

It seems that querying databases and analyzing such obtained data are becoming two completely separated tasks. Namely databases are more and more frequently offered through the Internet or other networks, and the remote user can query them with the provided queries, but any further manipulation with the retrieved data must be done on his local machine. The owner of the database simply is not interested in allowing remote users to perform time and memory consuming computations on his machine — he offers the data and takes care, that they are evaluated in the most effective way. And last but not least, the security factor is also important here.

Let us look at consequences which result from this separation.

First of all, the abilities of the end user are limited not directly by the expressive power of the query language, but by the expressive power of the *reflexive programming in the query language,* i.e., by abilities of application programs which may dynamically create and evaluate queries during computation. Therefore query languages whose expressive powers differ in the plain sense may appear to be equivalent in our scenario. If this happens to the recursive and nonrecursive queries in some language, we can legitimately say that the external computational power can compensate for the lack of recursion in the query language.

The answer to the above question can a priori depend on the complexity restrictions we impose on the programs. Further, the computational cost of the end user does not depend directly on the computational complexity of queries he issues any more. Perhaps the network transmission delay can be much more important factor for him, or waiting time to get his query evaluated because of many users querying the same data at the same time — in these cases the number of used queries becomes the main complexity measure. And certainly what remains important for him is the complexity of analyzing the data he gets.

1.2 Our results

First of all, we show that the question whether the lack of recursion in L can be compensated for by external computational power is generally equivalent to the old-fashioned logical question, whether complete L-theories are finitely axiomatizable or not and whether these axiomatizations can be effectively found.

We study in more detail two query languages, which have finite models with non-finitely-axiomatizable theories, namely Datalog with k variables, inequality and negation of edb's $\mathrm{Datalog}^k(\neq, \neg)$ and its nonrecursive fragment $\mathrm{FO}^k(\exists)$, which is (equivalent to) the existential fragment of first order logic with k variables. We show that one cannot emulate the recursion present in $\mathrm{Datalog}^k(\neq, \neg)$ at all, even using unlimited computational power on the application level, but with only $\mathrm{FO}^k(\exists)$ queries.

Then we attempt to measure, how much logics on two levels of this hierarchy differ, i.e., how many sentences of the stronger logic have to be evaluated, in addition to the sentences of the weaker one, to compute anything that can be computed with the unlimited access to queries of the stronger language. We prove that any function computable in polynomial time using $\mathrm{Datalog}^k(\neq, \neg)$ queries can be still computed in polynomial time using only $\log_2 \log_2 n$ such recursive queries, the rest of them being replaced by nonrecursive ones, and that there are functions, which require that many recursive queries. We show also that this bound is optimal, up to a constant factor, whose decreasing however costs increasing the degree of the polynomial bounding computation time. Therefore for structures of 1 Terabyte ($= 10^{12}$) elements 7 recursive queries are necessary and already suffice.

2 Preliminaries

The space limitations do not allow us to define all the notions we are going to investigate or use in this paper. The general reference texts which cover everything what is necessary to read this paper are Abiteboul, Hull and Vianu [1] and Ebbinghaus and Flum [5]. In fact, each of these books alone already does it.

2.1 L-generic functions

Let L be a query language. We call two structures $\mathbb{A}$ and $\mathbb{B}$ L-equivalent, $\mathbb{A} \equiv_L \mathbb{B}$ in symbols, if for every sentence $\varphi \in L$ holds $\mathbb{A} \models \varphi \Leftrightarrow \mathbb{B} \models \varphi$.

A function from the set Fin of all finite structures into $\mathbb{N}$ is called *L-generic* iff for every two L-equivalent structures $\mathbb{A}$ and $\mathbb{B}$ holds $f(\mathbb{A}) = f(\mathbb{B})$. In other words, $\ker f \supseteq \equiv_L$, i.e., equivalence classes of $\ker f$ are unions of equivalence classes of $\equiv_L$.

L-generic functions provide a generalization of the notion L-generic definability of classes of finite structures, which can be seen as considering L-generic functions into $\{0, 1\}$.

An L-generic function $f : \mathrm{Fin} \to \mathbb{N}$ is called an *L-bijection* iff f is surjective and $\ker f = \equiv_L$. f can be naturally considered as a bijective mapping of $\equiv_L$-equivalence classes into $\mathbb{N}$, from which the name comes.

2.2 L-generic computation model

We define now the *reflective relational machines* in the sense of Abiteboul, Papadimitriou and Vianu [2], which is in turn an extension of the model of *loosely coupled relational machines* of Abiteboul and Vianu [3].

Let L be a query language, closed under query composition. A *reflective relational L-machine* ($\mathrm{RM}(L)$ machine for short) is defined as follows.

It consists of a standard deterministic Turing machine component, including finite control and several work tapes. In addition it has a *relational store*, consisting of infinitely many relations R_i, $i = 1, 2, \ldots$ of fixed arities over some fixed finite set. These relations are provided as an input for the machine. One of the work tapes is distinguished to be the *L-query tape*, on which the machine can write arbitrary L-query of any finite subsignature of $\langle R_1, R_2, \ldots \rangle$, followed by # and a natural number. Upon entering a special state *ex* the query is evaluated in one step on the actual contents of the store, and the result is placed in the relation whose number is the second part of contents of the query tape. In case the second number is 0, the query on the tape should be a sentence. Then the output (*true* or *false*), which is obtained analogously in one step, is used to determine the next state of the machine, along with the symbols seen by heads on the standard tapes.

The initial configuration of the machine is as usually, where the relational store contains the input: it can be an arbitrary finite structure $\mathbb{A}$ of a fixed signature $\varrho = \langle R_1, \ldots, R_m \rangle$. The remaining relations $R_{m+1}, R_{m+2}, \ldots$ are set initially empty and will play the rôle of work registers.

We can now easily define, what it means, that a reflexive L-machine M computes a total function $f : \mathrm{Fin}(\varrho) \to \mathbb{N}$, or a total query $q : \mathrm{Fin}(\varrho) \to \mathrm{Fin}$.

It is possible to extend the above definition to query languages which are not closed under query composition. In this case we have to adopt additional restriction in the RM(L) model to make sure, that the machine cannot overcome limitations of the query language. One obvious and always applicable restriction is to forbid the use on non-Boolean queries. In some other context there are, however, other restrictions which assure us of this. In this paper we assume by the Boolean restriction only.

3 The Hierarchies in General

It is clear from the definition, that any function computable by a RM(L) machine is L-generic and that they are recursive in the standard sense. Our main goal in the paper is to investigate limitations concerning computability of L-generic functions by such machines. In a sense the maximal computational power is achieved when there is an RM(L)-computable L-bijection f. Then the ability to compute any given L-generic function g by an RM(L)-machine is just a recursion-theoretic question, namely whether there is a function $h : \mathbb{N} \to \mathbb{N}$ such that $g = h \circ f$.

It may happen, however, that for any an RM(L)-computable function f holds $\ker f \supsetneq \equiv_L$. In this case some recursive and L-generic functions remain noncomputable for RM(L), even if at the same time $\equiv_L$ is decidable. (In the other case that $\equiv_L$ is undecidable it is trivial and thus uninteresting.) If it is so, then it is possible that some extension of $L' \supseteq L$ with the same indistinguishability relation is stronger in the sense, that RM(L') machines can compute more functions than RM(L) machines.

First of all, we isolate the main reason for it, being that L-theories of some finite structures are not finitely axiomatizable in L, or that these axiomatizations exist but are not effectively computable.

Further we consider naturally occurring recursion-theoretic and complexity-theoretic hierarchies within extensions of L sharing the same indistinguishability relation $\equiv_L$.

3.1 Fine hierarchies of generic computation

We define as follows:

Definition 1. If L, L' are two query languages, then we write $L \preccurlyeq L'$ if every $\mathrm{RM}(L)$-computable function $\mathrm{Fin} \to \mathbb{N}$ is $\mathrm{RM}(L')$-computable, as well. If the inverse relation holds too we write $L \approx L'$; if it does not we write $L \prec L'$.

This opens us the possibility to investigate hierarchies of query languages w.r.t. $\preccurlyeq$. We can distinguish two types of hierarchies: those where inequalities follow from strict inclusions between indistinguishability relations, and *fine hierarchies,* which stratify query languages sharing the same indistinguishability relation. The fine hierarchies are the topic of this paper.

First the question, if any fine hierarchies at all exist, and if so, when, should be answered.

Definition 2. We say that the L-theory of a structure $\mathbb{A}$ is *finitely axiomatizable in* L iff there is a finite $T \subseteq L$ such that for any finite structure $\mathbb{B}$ holds $\mathbb{A} \equiv_L \mathbb{B}$ whenever the equivalence $\mathbb{A} \models \varphi \Leftrightarrow \mathbb{B} \models \varphi$ holds for every $\varphi \in T$.

If this is the case, we say that T *provides a finite axiomatization for* $\mathbb{A}$. Note that we do not require $\mathbb{A} \models T$, so T as such is not an axiomatization of the L-theory of $\mathbb{A}$. Only T together with the pattern of truth values of the sentences from T in $\mathbb{A}$ is such an axiomatization. We choose this definition because not all of the logics we are going to deal with are closed under negation.

First we analyze the connections between the existence of fine hierarchies and the existence of finite axiomatizations of L-theories of finite structures.

Theorem 3. *Let L be a query language, such that the relation $\models$ between L-sentences and finite models is recursive and $\equiv_L$ is recursive.*

Then every recursive L-generic function is computable by some $\mathrm{RM}(L)$ *iff the following two conditions are satisfied:*

FA1 *L-theory of each finite model is finitely axiomatizable.*
FA2 *There exists a recursive function FA from* Fin *into finite subsets of L such that $FA(\mathbb{A})$ provides a finite axiomatization for $\mathbb{A}$, for each* $\mathbb{A} \in \mathrm{Fin}$.

Proof. ($\Leftarrow$) We will show that there exists an L-bijection $f : \mathrm{Fin} \to \mathbb{N}$, computable by some $\mathrm{RM}(L)$ machine M.

Let $\mathbb{A}_1, \mathbb{A}_2, \ldots$ be any recursive enumeration of Fin.

The machine M we need proceeds as follows. Given $\mathbb{A}$ as input, M starts enumerating (encodings of) finite structures $\mathbb{A}_i$ for $i = 1, 2, \ldots$. For each enumerated structure it computes the set T_i of L-sentences providing a finite axiomatization for $\mathbb{A}_i$, queries $\mathbb{A}$ and $\mathbb{A}_i$ with all queries in T_i ($\mathbb{A}$ is queried by queries written on the L-tape, while for $\mathbb{A}_i$ it is achieved by computing on encodings, done entirely on the work tapes) and compares the results, determining thereby whether $\mathbb{A} \equiv_L \mathbb{A}_i$. It stops this procedure when the first $\mathbb{A}_k$ is found for which this equivalence holds. Certainly such $\mathbb{A}_k$ will be eventually found, because $\mathbb{A}$

itself must appear somewhere in the enumeration of finite structures. Note that outputting k at this moment already yields a function with kernel equal to $\equiv_L$, which need not be surjective however. In order to remedy this, M determines which structures among $\mathbb{A}_1, \ldots, \mathbb{A}_{k-1}$ are $\equiv_L$-equivalent — it is possible either using sets T_i or the standard algorithm for $\equiv_L$. The output of M is then the number of distinct $\equiv_L$-equivalence classes represented among $\mathbb{A}_1, \ldots, \mathbb{A}_{k-1}$. As it is easy to verify, this is already an L-bijection.

($\Rightarrow$) The other direction we begin with a preliminary fact.

Lemma 4. *If the relation $\equiv_L$ is recursive, then there exists a recursive L-bijection.*

Proof. The proof is analogous to the just presented proof of ($\Leftarrow$) of the Theorem. The only difference is that the device computing the L-bijection is a standard Turing machine, and its input $\mathbb{A}$ is given as encoding. Therefore it does not need the sets T_i to determine $\equiv_L$-equivalence of the input with other structures, because the algorithm for $\equiv_L$ suffices. □

Turning back to the proof of the Theorem, $\equiv_L$ is recursive, so by Lemma 4 there exists a recursive L-bijection f, which is in turn computable by some RM(L) machine M, by our assumption. Let $\mathbb{A}$ be a finite structure. Consider the computation of M on $\mathbb{A}$. The set of queries used by M in this computation provides a finite axiomatization for $\mathbb{A}$. Indeed, if there was another structure $\mathbb{B}$ with $\mathbb{A} \not\equiv_L \mathbb{B}$, but in which these queries gave the same results, then the computation of M in $\mathbb{B}$ would be identical as in $\mathbb{A}$ and would therefore result in the same output $f(\mathbb{A})$, thus contradicting L-bijectivity of f.

The above procedure can be used to determine recursively sets of sentences providing finite axiomatizations for finite structures, required in FA2. □

It may seem strange, but FA1 does not imply FA2 in general. A suitable example will be presented in the full version of the paper.

3.2 Examples

A natural example of the finitely axiomatizable case, where the fine hierarchy collapses, is the well-known hierarchy of logics $\mathrm{FO}^k \subseteq \mathrm{LFP}^k \subseteq \mathrm{PFP}^k \subseteq \mathrm{L}^k_{\infty\omega}$, in which all members share the same indistinguishability relation. Since it is known, that FO^k-theories of finite models are finitely axiomatizable by so-called Scott sentences (see Dawar, Lindell and Weinstein [4]), and since they can be effectively determined, it follows that among all recursive extensions of FO^k within $\mathrm{L}^k_{\infty\omega}$ the fine generic computation hierarchy collapses, which has been indeed known since the work of Abiteboul and Vianu [3]. There are other proofs of this fact, which do not refer to finite axiomatizability directly, like e.g. in author's [9]. The conclusion is that the lack of recursion on the level of query language (FO^k) can be substituted by computing power on the level of the application program.

Another example is the existential first order logic $\mathrm{FO}(\exists)$. As noted in [9], the indistinguishability relation of this logic is the isomorphism relation, finite models have always finitely axiomatizable theories, and these axiomatizations are effectively computable in the sense of FA2 in Theorem 3. Indeed, the set providing a finite axiomatization for $\mathbb{A}$ can be chosen to consist of two sentences: one of them asserts that $\mathbb{A}$ is a substructure of the structure at hand (and should be true), and the second asserts that there are at least $|\mathbb{A}|+1$ elements in the structure at hand (and should be false). It follows that any recursive isomorphism-invariant function is RM(FO($\exists$))-computable. In particular, the lack of recursion in this logic can be fully compensated for by the computational power outside of the query language. Another and interesting questions is whether it can be done efficiently. It can be shown that, e.g., any RM(FO($\exists$)) machine computing the transitive closure query must use at least $\frac{1}{2}\log n$ variables in structures of cardinality n, while in Datalog just three variables suffice.

3.3 Complexity of L-generic functions

We introduce the notions of complexity of L-generic functions computed by RM(L) machines as follows:

Definition 5. Let $K \subseteq L$ be two query languages with the same indistinguishability relation, and let $f : \mathrm{Fin} \to \mathbb{N}$ be an L-generic function.

$f \in \mathrm{TIME}_L(t(n))$ iff there is an RM(L) machine computing f and using at most $t(n)$ time for input structures of cardinality n. Evaluation of a query costs a unit time.

$f \in L\text{-QUERY}_K(q(n))$ iff there is an RM(L) machine computing f and querying its input at most $q(n)$ times with queries from $L \backslash K$, for input structures of cardinality n.

Complexity classes, in which functions obey simultaneously more resource bounds are defined as usual. E.g., $f \in [L\text{-QUERY}_K(q(n)), \mathrm{TIME}(t(n))]$ iff there is an RM(L) machine computing f and using at most $q(n)$ queries from $L \setminus K$ and time at most $t(n)$ for input structures of cardinality n.

Notions like e.g. PTIME_L are defined in the obvious way. $\mathrm{TIME}_L(\infty)$ will be used to denote the set of all total functions computable by RM(L) machines.

Now we can show that in reasonable cases fine hierarchies of generic computation have always a top element, which is moreover easily accessible from lower levels.

Proposition 6. *Let L be any query language such that $\equiv_L$ is decidable. Then there exists a query language $L^\top \supseteq L$ such that $\equiv_{L^\top} = \equiv_L$ and all recursive L-generic functions are computable by* RM($L^\top$).

Moreover, for any recursive, nondecreasing and unbounded function $f : \mathbb{N} \to \mathbb{N}$ holds

$$\mathrm{TIME}_{L^\top}(\infty) = L^\top\text{-QUERY}_L(f(n)).$$

Proof. Fix any recursive L-bijection h, which exists by Lemma 4. Now let $h^\top : \mathrm{Fin} \to \{0,1\}^\omega$ be defined by $h^\top(\mathbb{A}) = 1^{h(\mathbb{A})}0^\omega$.

Now let $L^\top = L \cup L'$, where L' consists entirely of Boolean queries φ_i, defined by

$$\varphi_i(\mathbb{A}) = \text{the } i\text{-th bit of } h^\top(\mathbb{A}).$$

It is immediate to verify that $\equiv_{L^\top} = \equiv_L$.

Let us now verify, that there is an $\mathrm{RM}(L^\top)$-machine M, which can compute our L-bijection h using only few $L^\top \setminus L$ queries.

For this purpose let $F : \mathbb{N} \to \mathbb{N}$ be an arbitrarily fast growing recursive function. The machine M we need queries its input structure $\mathbb{A}$ with $\varphi_{F(i)}$ for $i = 1, 2, \ldots$ until it finds i such that the query is false. In virtue of the definition of $h^\top$ this implies that $h(\mathbb{A}) < F(i)$. Then the machine, working all the time on its work tapes, finds by brute force search representatives of all $\equiv_L$-classes, whose h-values are at most $F(i) - 1$. It is an effective procedure, because h is surjective and we compute as long as all of the values appear. Then, having already all the representatives, M starts evaluating all L-sentences in them, until it finds a finite subset $K \subseteq L$ such that all these representatives have different K-theories. Finally, it queries its input structure $\mathbb{A}$ with all sentences of K, determining thereby a representative $\mathbb{B}$ with $\mathbb{B} \equiv_L \mathbb{A}$, and then outputs $h(\mathbb{B})$, which is by the definition of an L-bijection equal to $h(\mathbb{A})$.

It is easy to see, that choosing F growing fast enough, we can assure that the number of queries from $L^\top \setminus L$ necessary in this computation is smaller than any given recursive, nondecreasing and unbounded function of the cardinality of the input. $\square$

Note that the proof suggests a kind of trade-off between $L^\top$-QUERY_L complexity and the standard computational complexity. Namely, the faster F grows, the less queries from $L^\top \setminus L$ are needed, but more time and space is necessary in the remaining computation. E.g., when F is exponential and $h(\mathbb{A})$ happens to be only a little bit larger than $F(m) - 1$, then the number of structures considered in the computation will be exponentially larger than necessary. Moreover, the process of creating the set K is exponential in the number of structures to be considered itself.

Observe, however, that the simple algorithm, just querying the consecutive bits of $h^\top(\mathbb{A})$ works in time $O(h(\mathbb{A})) \cdot T_h(n))$, where $T_h(n)$ is the complexity of evaluating h. So in this case the overhead is not that bad.

We can show a lower bound, as well.

Proposition 7. *Suppose that every finite consistent set of L-sentences has infinitely many different extensions to a complete theory, i.e., to an L-theory of a finite structure. (Quite formally, for every $\mathbb{A}$ and every finite T there is a finite structure $\mathbb{B} \not\equiv_L \mathbb{A}$ such that $\mathbb{A} \models \varphi \Leftrightarrow \mathbb{B} \models \varphi$ for all $\varphi \in T$.)*

Let $L' \supsetneq L$ be such that L' and L have the same indistinguishability relation and any recursive L-generic function is $\mathrm{RM}(L')$-computable. Then there exist recursive L-generic functions such that any $\mathrm{RM}(L')$ machine computing them must use unbounded number of queries from $L' \setminus L$ during the computation.

Proof. Let f be any recursive L-bijection, which exists by Lemma 4. By our assumption, f is $\mathrm{RM}(L')$-computable. Suppose to the contrary, that the there is an $\mathrm{RM}(L')$ machine computing this function using at most m queries from $L' \setminus L$ for any input.

Then there are finitely many $\mathrm{RM}(L)$ machines $M_1, \dots, M_w$ such that for every $\mathbb{A}$ holds $M_i(\mathbb{A}) = f(\mathbb{A})$ for at least one $i \leq w$. Indeed, we obtain M_i as follows: they simulate M, and, whenever M attempts to evaluate in the input a sentence not in L, M_i supplies it with the answer, which is encoded in its finite control. The answers are organized in a list, and are assigned to non-L queries in the order of appearance in the computation of M. So we need $w = 2^m$ such machines, one for each possible sequence of m results of evaluations of $L' \setminus L$ queries. Certainly most of the machines compute nonsense, but at least one provides the true answers, and therefore computes the correct output.

Let $\mathbb{A}_1$ be any finite model. At least one of the machines, w.l.o.g. M_1, when run on $\mathbb{A}_1$, stops and outputs $f(\mathbb{A}_1)$. But M_1 uses only a finitely many L-sentences in this computation, and there must be a structure $\mathbb{A}_2 \not\equiv_L \mathbb{A}_1$ such that all sentences evaluated by M_1 have the same truth values in $\mathbb{A}_2$ as in $\mathbb{A}_1$. So M_1 run on $\mathbb{A}_2$ computes precisely as in $\mathbb{A}_1$ and outputs $f(\mathbb{A}_1)$, thus making an error. So there must be another machine, w.l.o.g. M_2, which run on $\mathbb{A}_2$ halts and outputs $f(\mathbb{A}_2)$. But then similarly there must exist $\mathbb{A}_3$, $\equiv_L$-inequivalent to both $\mathbb{A}_1$ and $\mathbb{A}_2$, but such that $M_1(\mathbb{A}_3) = f(\mathbb{A}_1)$ and $M_2(\mathbb{A}_3) = f(\mathbb{A}_2)$. Therefore there must be a third machine M_3 with $M_3(\mathbb{A}_3) = f(\mathbb{A}_3)$, and so on. Continuing in this manner, we easily find that there exists an input structure on which all the machines $M_1, \dots, M_w$ make an error, and thus arrive at a contradiction, finishing our proof. □

4 Existential First Order Logic and Datalog

In this section we are going to investigate two natural and widely studied query languages, namely Datalog with negation of edb's and inequalities and its recursion-free fragment, i.e., existential first order logic. Due to reasons, which are explained in §3.2, we consider the k-variable fragments of these query languages, denoted respectively by $\mathrm{Datalog}^k(\neq, \neg)$ and by $\mathrm{FO}^k(\exists)$. Their indistinguishability relations are equal and decidable, even in PTIME, as it has been shown by Kolaitis and Vardi [7].

Since the names of the two above query languages will appear very frequently in this section, we introduce abbreviations: $\mathrm{Datalog}^k(\neq, \neg)$ will appear as D^k and $\mathrm{FO}^k(\exists)$ as F^k.

The query languages we consider in this section are not closed under query composition, which is a consequence of the fact they are not closed under negation. On the other hand, unrestricted $\mathrm{RM}(L)$ machines can compose queries. Therefore we impose the universal restriction, namely to Boolean queries, which works for any query language.

4.1 Emergence of the hierarchy

It is known from the work of Rosen and Weinstein [8], that the hierarchy must exist, because there are finite models, whose theories in neither F^k nor D^k are finitely axiomatizable in the respective logics. We improve them a little bit to the form of the following.

Theorem 8. *For every $k > 2$ and every constant c holds*

$$\mathrm{D}^k\text{-QUERY}_{\mathrm{F}^k}(c) \subsetneq \mathrm{TIME}_{\mathrm{D}^k}(\infty), \tag{1}$$
$$(\mathrm{D}^k)^\top\text{-QUERY}_{\mathrm{D}^k}(c) \subsetneq \mathrm{TIME}_{(\mathrm{D}^k)^\top}(\infty). \tag{2}$$

Proof. The strictness of the first inclusion will follow from the next theorem.

The strictness of the second is a simple improvement of the argument of Rosen and Weinstein, so we leave it out, because of space limitations. □

In particular, (1) and (2) imply the existence of the hierarchy

$$\mathrm{F}^k \prec \mathrm{D}^k \prec (\mathrm{F}^k)^\top \approx (\mathrm{D}^k)^\top. \tag{3}$$

Now we establish precise distances between levels in this hierarchy, based on the number of queries, if no other complexity restrictions are imposed.

Theorem 9. *For any recursive, nondecreasing and unbounded function $f : \mathbb{N} \to \mathbb{N}$ holds:*

$$\mathrm{TIME}_{\mathrm{D}^k}(\infty) \subseteq \mathrm{D}^k\text{-QUERY}_{\mathrm{F}^k}(f(n)) \tag{4}$$
$$\mathrm{TIME}_{(\mathrm{F}^k)^\top}(\infty) \subseteq (\mathrm{F}^k)^\top\text{-QUERY}_{\mathrm{D}^k}(f(n)) \tag{5}$$

Proof. Inclusion (5) follows from the general fact. Inclusion (4) follows by an argument quite similar to that used in the proof of Theorem 10 below. It is therefore left out. □

4.2 Tradeoffs

As we have already noted after proof of Proposition 6, there seems to be a kind of tradeoff between complexity measured in terms of time and space and the complexity measured in terms of the number of queries. What we prove now, is that this tradeoff really exists, and the time price of decreasing the number of queries has to be paid. We do it analyzing the first strict inclusion in the hierarchy (3). The second strict inclusion there is much less interesting because $(\mathrm{F}^k)^\top$ appearing in it is a kind of an artificial closure operator, which is not even uniquely determined, and the author is unaware of any natural query language $\approx$-equivalent to it.

Theorem 10. *For every k holds*

$$\mathrm{PTIME}_{D^k} = [\mathrm{PTIME}_{\mathrm{D}_k}, \mathrm{D}^k\text{-QUERY}_{\mathrm{F}^k}(\log\log n)] \tag{6}$$

and for each $k > 2$ and each m there is a constant c such that

$$\mathrm{TIME}_{\mathrm{D}^k}(n^2) \nsubseteq [\mathrm{TIME}_{\mathrm{D}_k}(n^m), \mathrm{D}^k\text{-QUERY}_{\mathrm{F}^k}(c\log\log n)]. \tag{7}$$

In particular, if $\lim_{n\to\infty} q(n)/\log\log n = 0$, then

$$[\mathrm{PTIME}_{\mathrm{D}_k}, \mathrm{D}^k\text{-QUERY}_{\mathrm{F}^k}(q(n))] \subsetneq \mathrm{PTIME}_{\mathrm{D}^k}. \tag{8}$$

Proof. We begin with (6). Suppose that M is a $\mathrm{RM}(\mathrm{D}^k)$ and computes a function f, making at most n^m steps of computation in structures of cardinality n. W.l.o.g. we can assume $m > k > 2$.

We are going to construct an $\mathrm{RM}(\mathrm{D}^k)$ machine N witnessing that $f \in [\mathrm{PTIME}_{\mathrm{D}_k}, \mathrm{D}^k\text{-QUERY}_{\mathrm{F}^k}(\log\log n)]$. Generally it simulates the behaviour of M. It is equipped with a counter, which counts steps of computation of M. (Recall that queries are evaluated in one step of M, so we count one step for them in the simulation, whatever N really does emulating this action of M.) N has a special query buffer for storing D^k queries to be evaluated later.

The following pseudo-Pascal program represents the algorithm realized by N. The integer valued variable *max* is initialized to 17 and *counter* to 0.

begin
    do
        emulate and count consecutive steps of the computation of M, replacing each evaluation of a query $\varphi \in \mathrm{D}^k$ by evaluation of its *max*-step unwinding, which is in F^k; if this unwinding evaluates to *false*, then add the original query φ to the buffer
    until *counter* = *max* or M halts;
    evaluate the disjunction of all queries in the buffer;
    if the result is *false* and M halted
        then
            output the output of M and halt
        else
            empty the buffer;
            counter := 0;
            $max := max^m$;
    endif;
goto begin;

First let us see, that N is total and computes the same function as M. Indeed, suppose that $|\mathbb{A}| = n$. Then the computation of M in $\mathbb{A}$ takes at most n^m steps, and the deepest induction in a D^k query used in this computation requires at most $n^k \leq n^m$ steps. Since each execution of all statements in the outer **goto**-loop causes *max* to increase, eventually the value of *max* is at least n^m. Then in the next full emulation of M all approximations of D^k queries by F^k queries evaluate to correct answers, and M simulated with these results halts within *max* steps. Moreover, the disjunction of all queries stored in the buffer must evaluate to *false*, and consequently N halts and outputs what M does, which is the correct value. On the other hand, the only way N can halt is when M halts in the last simulation and all queries in this simulation evaluate to their true results, which in virtue of the above analysis leads to the correct output.

Now let us consider the complexity of N. Note that each cycle of emulation, including tests and resetting of buffer and integer variables, requires time polynomial (of some fixed degree) in the value of *max* and precisely one $\mathrm{D}^k \setminus \mathrm{F}^k$ query.

The value of *max* is never greater than n^{2m}. Indeed, any complete simulation performed with $max \geq n^m$ ends by outputting the value of $f(\mathbb{A})$ and halting. So in the worst case the last unsuccessful emulation is performed with $max = n^m - 1$, and then the next one, with $max = (n^m - 1)^m < n^{2m}$, already halts.

Finally, the outer **goto** loop is executed at most as many times as there are necessary to achieve n^{2m}, starting from 17 and rising the value each time to the m-th power. This number is easily seen to be at most $\log\log n$. This provides us the last two facts we need: the number of D^k queries is at most $\log\log n$, and the running time of N is bounded by $p(n) \cdot \log\log n$, for some fixed polynomial p. Therefore $f \in [\mathrm{PTIME}_{\mathrm{D}_k}, \mathrm{D}^k\text{-QUERY}_{\mathrm{F}^k}(\log\log n)]$.

The proof of (7) will be sketched only. We begin it with some preliminaries: The set of all satisfiable F^k sentences of any fixed vocabulary has a model, and even a finite model. Namely, every structure of this vocabulary, which has a substructure satisfying the *extension axiom* ϵ_k *with* k *variables* satisfies all satisfiable F^k sentences. Such structures exist in a great number, e.g., a randomly chosen directed graph of size 2^{k^2} already satisfies ϵ_k (for directed graphs) with probability very close to 1. See Rosen and Weinstein [8].

Let us fix a sequence $w = w_1 w_2 \dots w_l \in (\mathbb{N} \cup \{\infty\})^*$. We construct a structure $\mathbb{A}(w)$ as follows: We begin with two disjoint directed chains (included in unary relations E_0, E_1, respectively) of length $|w|$. The beginning point of E_0 is marked by a constant b_0, and that of E_1 by b_1.

We fix a finite directed graph $G \models \epsilon_{k+1}$ and one of its vertices. The choice of both is unimportant. Let g denote the cardinality of the vertex set of G.

Now we add to the two-chain graph we have had $2|w|$ disjoint copies of G, identifying the distinguished vertices in the copies with elements in the chains.

Finally for each $i \leq |w|$, if $w_i \in \mathbb{N}$, we connect the i-th elements in E_0 and E_1 by a chain made of w_i new vertices; otherwise we do not add anything.

Let for $w \in (\mathbb{N} \cup \{\infty\})^*$ the string $\hat{w} \in \{0,1\}^*$ be defined as $\hat{w}_1 \hat{w}_2 \dots \hat{w}_l$, where $\hat{w}_i = 1$ iff $w_1 \in \mathbb{N}$.

Paths of finite length joining vertices in $\mathbb{A}(w)$ mark bits of $\hat{w}$, like on Fig. 1 below.

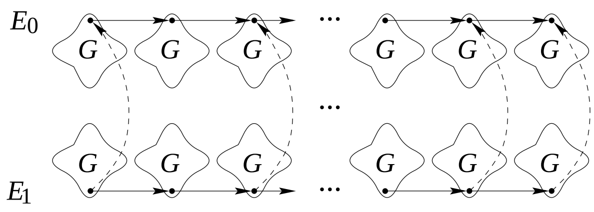

Fig. 1. An example of a structure $\mathbb{A}(w)$. In this case $\hat{w} = 101\ldots011$.

We assume the following lemma without proof.

Lemma 11. *Let $w \in (\mathbb{N} \cup \{\infty\})^*$ and let w' be obtained from w by replacing an occurrence of ∞ by $s \in \mathbb{N}$.*

Then $\mathbb{A}(w)$ and $A(w')$ are indistinguishable by sentences $\varphi \in \mathrm{F}^k$ with less than s quantifiers, i.e., for all such sentences φ holds $\mathbb{A}(w) \models \varphi \Leftrightarrow \mathbb{A}(w') \models \varphi$. □

The function we are interested in is defined by $f : \mathbb{A}(w) \mapsto \hat{w}$. (It is not too difficult to extend it to a function defined for all finite structures and $\mathrm{RM}(\mathrm{D}^k)$-computable.)

Suppose M is an $\mathrm{RM}(\mathrm{D}^k)$ machine computing our function in polynomial time, say n^m. Let $q_M(n)$ denote the D^k-$\mathrm{QUERY}_{\mathrm{F}^k}$ complexity of M.

[1] We begin with $l = 10$, say, choose $w = \infty^{10}$, construct the structure $\mathbb{B}_0 = \mathbb{A}(\infty^{10})$ and run M on this structure. In this computation M must use at least one $\mathrm{D}^k \setminus \mathrm{F}^k$ query. Indeed, suppose to the contrary that M uses only F^k queries. Due to the time bound, their length, and therefore the number of quantifiers in each of them as well, is bounded by $s_0 = (20g)^m$. Then M run on $\mathbb{B}_1 = \mathbb{A}((s_0 + 1)\infty^9)$ has during the whole computation identical answers of queries as in the computation on $\mathbb{B}_0$ by Lemma 11, and therefore it halts and outputs 0^{10}, which is a contradiction with assumption that it computes f.

We get $q_M(20g)) \geq 1$.

[2] Let us now consider the computation of M on $\mathbb{B}_1$. We claim M must use at least two $\mathrm{D}^k \setminus \mathrm{F}^k$ queries.

Indeed, suppose to the contrary that M uses only one $\mathrm{D}^k \setminus \mathrm{F}^k$ query and finishes its computations with the right answer. Certainly the use of this query is forced by identity of computations of M on our input and on $\mathbb{B}_0$ up to the point of using the first $\mathrm{D}^k \setminus \mathrm{F}^k$-query.

This time the computation uses at most $s_1 = (20g + s_0 + 1)^m$ time, and so is the maximal possible number of quantifiers in the used F^k queries. Let us see what happens to the computation of M on $\mathbb{B}_2 = \mathbb{A}((s_0+1)(s_1+1)\infty^8)$. Again M uses in this computation its first $\mathrm{D}^k \setminus \mathrm{F}^k$ query exactly as in the computation on $\mathbb{B}_0$. The answer must be different as in $\mathbb{B}_0$, however, because otherwise M would have halted and answered 0^{10}. So the answer is identical as in $\mathbb{B}_1$, and, since all the remaining queries used by M in $\mathbb{B}_1$ are from F^k, and moreover $\mathbb{B}_1$ and $\mathbb{B}_2$ do not differ w.r.t. F^k queries with less than s_1+1 quantifiers, which follows from Lemma 11, the machine halts and answers 10^9, which yields a contradiction.

So $q_M(20g+(20g)^m+1)\geq 2$.

[3] We consider the computation of M on $\mathbb{B}_2$. We claim this computation must use at least three $\mathrm{D}^k\setminus\mathrm{F}^k$ queries. Indeed, suppose to the contrary it uses only two queries. The first of them is caused by the computation on $\mathbb{B}_0$ and the second by the computation on $\mathbb{B}_1$. Similarly as before, the first query must give the same result in $\mathbb{B}_1$ and in $\mathbb{B}_2$ (the opposite of the result in $\mathbb{B}_0$), so the results of the second one must be different, to prevent M from answering 10^9 in $\mathbb{B}_2$. Let $s_2=(20g+s_0+1+s_1+1)^m$. Now we consider $\mathbb{B}_3=\mathbb{A}((s_0+1)(s_1+1)(s_2+1)\infty^7)$ and the computation of M on this structure. And again, until a third $\mathrm{D}^k\setminus\mathrm{F}^k$ query is used, this computation must be identical as in $\mathbb{B}_2$, and since the remaining computation of M in $\mathbb{B}_2$ uses only F^k queries and they give in $\mathbb{B}_3$ the same results (by Lemma 11), actually the third $\mathrm{D}^k\setminus\mathrm{F}^k$ query is never used and M halts in $\mathbb{B}_3$ answering 110^8, which yields a contradiction.

So $q_M(20g+(20g)^m+1+(20g+(20g)^m+1)^m+1)\geq 3$.

[...] Continuing in the same manner we get finally $q_M(n_i)\geq i$ for each $i\leq 9$, where $n_1=20g$ and $n_{i+1}=n_i+n_i^m+1\leq(n_i+1)^m$. It is elementary that $n_i\leq(40g)^{m^{i-1}}$.

Then we do not have any more place to play and we have to increase the length of the initial word, to, say, 100. Repeating the same construction we conclude $q_M(n_i)\geq i$ for each $i\leq 99$, where $n_1=100g$ and $n_{i+1}=n_i+n_i^m+1\leq(n_i+1)^m$. It is again elementary that $n_i\leq(400g)^{m^{i-1}}$.

In total this makes $q_M(n_i)\geq i$ for some $n_i\leq(4(i+1)g)^{m^{i-1}}$, for every i.

It follows by a simple computation that $q_M(n)\geq c\log\log n$ for infinitely many n and some constant c, which depends on m solely.

The construction of an $\mathrm{RM}(\mathrm{D}^k)$ machine computing f in quadratic time is left for the reader as an exercise. □

There is an interesting methodological consequence of the results we have proven. It is a kind of common opinion that pebble games and computational complexity are the main determinants of the expressiveness of query languages. And to some extent it is so indeed: in order to verify whether a given class of graphs is definable in, say, LFP, one has to check (1) whether it is compatible with $\equiv_{\mathrm{FO}^k}$ for some k, and for this task suitable games are sound and complete; and (2) whether its time complexity w.r.t. $\mathrm{RM}(\mathrm{FO}^k)$ is within PTIME in appropriate sense. It has been shown by Abiteboul and Vianu [3] that these two conditions are necessary and already suffice.

In our case a pebble game appropriate for Datalog has been found by Kolaitis and Vardi [7]. But now there is a life beyond the games. We have just found functions which are in PTIME and are F^k-generic, but which remain noncomputable by $\mathrm{RM}(\mathrm{F}^k)$ machines, no matter how much time and space they are allowed to use. It seems therefore that the games for Datalog and existential first order logic are a little bit too strong when compared to the logics.

5 Conclusions

In this paper we have started investigating whether the lack of recursion in a query language can be compensated for by the computational power outside of the query language—namely by replacing a recursive query by a sequence of nonrecursive queries plus an analysis of such obtained data. We believe that this question can soon become of practical relevance, corresponding to the situation of a user querying a database over the net.

Using reflexive relational machines, computing functions from finite structures into natural numbers as a computation model, we have shown that the answer is positive iff complete theories in the query language are finitely axiomatizable and these finite axiomatizations are effectively computable.

Then we have investigated in more depth the non-finitely axiomatizable situation encountered in Datalog with bounded number of variables, enriched by inequality and negation of edb's. We have shown that in this case every function computable in PTIME with unlimited access to recursive queries can be computed using only $\log\log n$ recursive queries, the rest of them being replaced by nonrecursive ones. Moreover, this result is optimal, because there are functions which require this number of recursive queries.

Acknowledgments. The author would like to express his sincere thanks to Eric Rosen, who, among other valuable comments, has suggested the use of Lemma 11 in the proof of Theorem 10. The anonymous referees provided constructive criticism, which have helped me to improve the presentation of my results.

References

1. S. Abiteboul, J. Hull and V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.
2. S. Abiteboul, C. Papadimitriou and V. Vianu, The power of reflective relational machines, in: *Proc. 9th Symposium on Logic in Computer Science,* 1994, pp. 230–240.
3. S. Abiteboul and V. Vianu, Generic computation and its complexity, in: *Proc. ACM SIGACT Symp. on the Theory of Computing,* 1991, pp. 209–219.
4. A. Dawar, S. Lindell and S. Weinstein, Infinitary logic and inductive definability over finite structures, *Information and Computation*, **119**, 1995.
5. H.-D. Ebbinghaus and J. Flum, *Finite Model Theory*, Springer Verlag, 1995.
6. H. Gaifman, On local and nonlocal properties, in: J. Stern (ed.), *Logic Colloquium '81*, North Holland, 1982, pp. 105–135.
7. Ph. Kolaitis and M. Vardi, On the expressive power of Datalog: tools and a case study, in: *Proceedings of the 9th Symposium on Principles of Database Systems,* 1990, pp. 46–57.
8. E. Rosen and S. Weinstein, Preservation theorems in finite model theory, in: D. Leivant (ed.), *Logic and Complexity,* Springer Verlag, 1995.
9. J. Tyszkiewicz, On the Kolmogorov expressive power of Boolean query languages, to appear in *Theoretical Computer Science.*
   Preliminary version appeared in: G. Gottlob, M.Y. Vardi (eds.), *Proc. ICDT'95,* Lecture Notes in Computer Science 893, Springer Verlag, pp. 97–110